

RandFile package for *Mathematica* for accessing file-based sources of randomness

J.A. Miszczak^{1,*} M. Wahl²

¹Institute of Theoretical and Applied Informatics, Polish Academy of Sciences,
Bałtycka 5, 44-100 Gliwice, Poland

²PicoQuant GmbH,
Rudower Chaussee 29 (IGZ), 12489 Berlin, Germany

12/02/2013 (v. 0.15)

After 40 years of development, one might think that the making of random numbers would be a mature and trouble-free technology, but it seems the creation of unpredictability is ever unpredictable.

B. Hayes [7]

Abstract

We present a package for *Mathematica* computer algebra system which allows the exploitation of local files as sources of random data. We provide the description of the package and illustrate its usage by showing some examples. We also compare the provided functionality with alternative sources of randomness, namely a built-in pseudo-random generator and the package for accessing hardware true random number generators.

1 Introduction

Random numbers are essential for solving a variety of physical and mathematical problems [5, 11] and the quality of random numbers used in simulations is critical for the outcomes [4, 3]. It is of no surprise that the considerable research effort has been devoted to the development of methods for generating pseudo-random [15, 10] and truly random numbers [19, 6, 9].

One of the main advantages of pseudo-random number generators is their ability to reproduce the sequence of random numbers used in a given simulation. However, one can argue if the amount of randomness provided by such generators is sufficient [4, 3]. On the other hand, random numbers generated using hardware generators (*eg.* quantum random number generators [17, 8]), provide fast sources of truly random numbers. In this case, however, the user has to have a specific hardware for their disposal and loses the ability to reproduce the sequence of random numbers used during the simulation. The ideal solution would allow the user to employ true random data sources without the need for a specific hardware and with the ability to repeat the experiments.

The presented work aims to fill this gap. We provide RandFile [12] package for *Mathematica* computer algebra system, which allows using use files with random data for generating

*E-mail: miszczak@iitis.pl

random numbers used in simulations [16] easily. There has been twofold motivation for the work on the provided package. Firstly, the presented package allows using fast, high-quality sources of randomness in simulations. Secondly, as with the help of the presented package one can use any file as a source of data for producing random numbers, this enables users to observe the influence of the quality of random data on the outcomes of conducted simulations.

This paper is organized as follows. In Section 2 we provide an overview of the presented package and describe its functionality. In Section 3 we compare the presented package with the alternative methods providing sources of randomness available for *Mathematica* users. In Section 4 we provide some usage examples. Finally, in Section 5 we provide the summary of the presented work and some concluding remarks.

2 RandFile package

Package `RandFile` is designed to provide a user interface for accessing random numbers generated using random binary data stored in files on a hard drive. In the following we assume that the user obtained as sample of random data which are stored in `random_file.bin` file.

2.1 Controlling the source of random data

Almost all functions provided by the package require setting a global variable pointing to the file with random data. This can be achieved by using `SetTrueRandomDataFile` function. For example

```
SetTrueRandomDataFile["/home/user/data/random_file.bin"]
```

The user should note that it is advised to use this function only once during a session. After the file has been set, it is possible to produce random numbers using one of the provided functions. For example, one can call

```
TrueRandomInteger[]
```

to obtain 0 or 1.

Some functions provided by the package accept an extra argument which can be used to assign the file which should be used during the call. Using this functionality the above result can be reproduced using the command

```
TrueRandomInteger["random_file.bin"]
```

assuming the used file is located in one of the directories included in `$Path` *Mathematica* variable.

If one intends to use `TrueRandomSequence` function they must declare the maximal sequence length using `SetMaxTrueRandomSequenceLength`. Random sequences are declared by using `TrueRandomSequence[n]` function. The declared sequences can be displayed by calling `GetTrueRandomDataMarkers[]`. Once defined, the used maximal length cannot be changed during the session.

The manipulation of the source of random data can be achieved using the following functions.

- `SetTrueRandomDataFile[fName]` sets the global variable `TrueRandomDataFile` used by the functions for generating random numbers. For example

```
SetTrueRandomDataFile["/home/user/data/random_file.bin"]
```

- `GetTrueRandomDataFile[]` displays the value of the global variable `TrueRandomDataFile`.

- **CloseTrueRandomDataFile**[] closes the file assigned by the global variable set using **SetTrueRandomDataFile**.
- **TrueRandomDataFile** is the global variable, defined in `RandFile`Private` context, storing a string with the name of the file containing random data.
- **TrueRandomDataFileBytesCount** is the global variable for storing a number of bytes in the **TrueRandomDataFile**.
- **BlockTrueRandom**[ex] evaluates expression `ex` and stores the starting position in the file **TrueRandomDataFile**. The position in the file is restored after the execution. Note that this function works only with expressions using the global variable for pointing to a file with random data.
- **TrueRandomSequence**[] changes the current position in the file assigned by global variable **TrueRandomDataFile**, so that the numbers generated after the execution of this function will not overlap with the previously generated numbers.
- **TrueRandomSequence**[pos] marks the current position in the file with random data or, if the position `pos` is already marked, returns to this position in the file.
- **TrueRandomDataMarkers** is a global variable for accumulating positions in the file with random data used by **TrueRandomSequence** function. Note that this array is sorted with respect to the second element.
- **SetMaxTrueRandomSequenceLength**[len] declares the maximal length, expressed in bytes, of the true random sequence used during the session.
- **GetMaxTrueRandomSequenceLength**[] displays the maximal length of the random sequence which can be used during the session.
- **GetTrueRandomDataMarkers**[] displays the currently set markers in the data file that are used by **TrueRandomSequence**.

2.2 Obtaining random numbers

Package `RandFile` aims to provide a user interface similar to the standard user interface for using random numbers provided by *Mathematica*. The user should be able to incorporate the package into existing simulations with minimal changes in the source code only.

2.2.1 Integer numbers

The package provides the following functions for producing random integer numbers.

- **TrueRandomInteger**[] produces an integer number in $[0, 1]$.
- **TrueRandomInteger**[{a,b}] produces an integer number in $[a, b]$.
- **TrueRandomInteger**[{a,b},n] produces n integer numbers in $[a, b]$.
- **TrueRandomInteger**[{a,b},{d₁,d₂,...,d_k}] produces a $\{d_1, d_2, \dots, d_k\}$ -dimensional array of integer numbers in $[a, b]$.

All of the above functions have their counterparts of the form

TrueRandomInteger[params, fName]

For example, the command

```
TrueRandomInteger[n]
```

will produce n one of 0 or 1 using a global file with random data, while using

```
TrueRandomInteger[n, "my_random_file.bin"]
```

will generate a similar result using data from file `my_random_file.bin`

From the above example one can see why the counterpart of the standard pattern of the form

```
RandomInteger[max]
```

is missing. This is motivated by the fact that if one needs to use specific files with random data during the execution, such pattern (**TrueRandomInteger**[max, fName]) would collide with the pattern for producing n random integer numbers (**TrueRandomInteger**[n, fName]).

The user should be aware of the fact that the amount of random bytes utilized by the functions for producing integer numbers depends on the range.

2.2.2 Real numbers

As in the case of integer numbers, the functions for producing real numbers use by default source of random numbers assigned in the global variable set by **SetTrueRandomDataFile**. To obtain random real numbers one can use one of the following functions.

- **TrueRandomReal**[] produces 32-bit real number in $[a, b]$.
- **TrueRandomReal**[{a,b}] produces 32-bit real number in $[a, b]$.
- **TrueRandomReal**[b] produces 32-bit real number in $[0, b]$.
- **TrueRandomReal**[{a,b}, n] produces n 32-bit real numbers in $[a, b]$.
- **TrueRandomReal**[{a,b}, {d₁, d₂, ..., d_k}] produces a {d₁, d₂, ..., d_k}-dimensional array of 32-bit real numbers in $[a, b]$.
- **TrueRandomReal**[**NormalDistribution**[μ, σ]] produces 32-bit real number sampled from **NormalDistribution**[μ, σ].
- **TrueRandomReal**[**NormalDistribution**[μ, σ], n] produces n 32-bit real numbers sampled from **NormalDistribution**[μ, σ].
- **TrueRandomReal**[**NormalDistribution**[μ, σ], {d₁, d₂, ..., d_k}] produces a {d₁, d₂, ..., d_k}-dimensional array of 32-bit real numbers sampled from **NormalDistribution**[μ, σ].

Also in this case all of the above functions have their counterparts of the form

```
TrueRandomReal[params, fName]
```

where `params` refers to the standard parameters, while `fName` can be used to assign a file with random data used during the execution. For example, using the command

```
TrueRandomReal[b]
```

will produce a real number in $[0, b]$ using a global file with random data, while using the command

```
TrueRandomReal[b, "my_random_file.bin"]
```

will generate a similar result using data from file `my_random_file.bin`

In contrast to the functions for generating random integers, the functions producing random real numbers require a fixed number of bytes during each call – by default each call requires 4 bytes and this value can be changed by altering **TrueRandomReal** function. This allows the anticipation of the size of the required data files or the length of sequences defined by **SetMaxTrueRandomSequenceLength** and **TrueRandomSequence** functions.

2.2.3 Complex numbers

The third category of functions provided by the package for producing random numbers can be used to obtain complex numbers. There are the following functions in this category

- **TrueRandomComplex**[] gives a true random complex number with real and imaginary parts in the range 0 to 1.
- **TrueRandomComplex**[{min,max}] gives a true random complex number in the rectangle with corners given by the complex numbers min and max.
- **TrueRandomComplex**[{min,max},n] gives a list of n true random complex numbers in a given rectangle.
- **TrueRandomComplex**[{min,max},{ d_1, d_2, \dots, d_k }] gives a $\{d_1, d_2, \dots, d_k\}$ -dimensional array of true random complex numbers in a given rectangle.

As in the case of integer and real numbers, the above functions can be called with the third optional argument for assigning the file used to obtained random data during the call.

2.2.4 Sampling functions

The package provides functions for obtaining weighted samples following the convention used by **RandomChoice** and **RandomSample** standard *Mathematica* functions.

- **TrueRandomChoice**[{ e_1, e_2, \dots, e_k }] gives a true random choice of one of $\{e_1, e_2, \dots, e_k\}$.
- **TrueRandomChoice**[{ e_1, e_2, \dots, e_k }, n] gives a list of n true random choices.
- **TrueRandomChoice**[elist,{ d_1, d_2, \dots, d_k }] gives a $\{d_1, d_2, \dots, d_k\}$ -dimensional array of true random choices.
- **TrueRandomChoice**[{ w_1, w_2, \dots, w_n } -> { e_1, e_2, \dots, e_n }] gives a true random choice weighted by w_i .
- **TrueRandomChoice**[{ w_1, w_2, \dots, w_n } -> { e_1, e_2, \dots, e_n }, k] gives k true random choices weighted by w_i .
- **TrueRandomChoice**[{ w_1, w_2, \dots, w_n } -> { e_1, e_2, \dots, e_n }, { d_1, d_2, \dots, d_k }] gives a $\{d_1, d_2, \dots, d_k\}$ -dimensional array of true random choices weighted by w_i .
- **TrueRandomSample**[l] gives a true random permutation of the list l .
- **TrueRandomSample**[l,n] gives n elements from the true random sample of the list l . Note that it is possible to take at most **Length**[l] elements.
- **TrueRandomSample**[{ w_1, w_2, \dots, w_n } -> { e_1, e_2, \dots, e_n }, k] gives k elements from the non-uniform sample of elements $\{e_1, e_2, \dots, e_n\}$ with weights $\{w_1, w_2, \dots, w_n\}$.

2.2.5 Deprecated **Random[]** interface

As of version 6 *Mathematica* function **Random[]** is deprecated and one is encourage to use functions **RandomInteger**, **RandomReal** and **RandomComplex** instead. However, for the sake of compatibility, RandFile package implements the counterpart of the standard **Random[]** function.

- **TrueRandom**[type, range] gives a true random number of type **Integer**, **Real** or **Complex** in a specified range.

The range specification follows the convention used in **Random[]** function and the user is advised to consult *Mathematica* documentation in this matter.

3 Comparison with other sources of randomness

The main source of random numbers provided by *Mathematica* is the built-in pseudo-random number generator. One of the alternative sources of randomness is provided by the TRQS package [13, 14], which allows accessing the quantum true random number generators, using an off-line [8], as well as an on-line, interface [1].

3.1 User interface

Package RandFile was designed for accessing sources of true random data in the form suitable for *Mathematica* users. For this reason the functions provided by the package follow the convention used in the standard functions as close as possible. The main differences between the standard functions and the functions implemented by the package are caused by the need for assigning and managing the access to the file with random data.

Table 1 provides a comparison of patterns defined by *Mathematica* with the patterns used by the package for the purpose of producing numbers in **Integer**, **Real** and **Complex** domains. As one can see, the functions provided by the package follow the standard interface in most of the cases. This allows the integration of the package with the existing *Mathematica* programs without any significant changes in the existing code.

The main difference in the user interface provided by RandFile package in comparison to the standard functions is that all functions can be used with an optional last argument for assigning a file with random data used during the call. This allows the control of the source of randomness and thanks to this using the disjoint sources during the subsequent calls. This functionality allows, when required, using different files for producing non-overlapping sequences of true random numbers. One should note, however, that this can be also achieved by using **TrueRandomSequence** function.

Among other noticeable differences one can point out the fact that the counterpart of **SeedRandom** function provided by the package is named **TrueRandomSequence**. This is motivated by the requirement of providing the non-overlapping sequence of random numbers while using the **TrueRandomSequence** function.

One should also notice that one pattern for generating integer numbers is missing. This is caused by the problem discussed in Section 2.2.1.

3.2 Package efficiency

One of the important characteristics of the random number generators is their speed. This characteristic is especially important in simulations requiring large amount of random data.

Standard <i>Mathematica</i> pattern	RandFile implementation
—	SetTrueRandomDataFile [fName]
BlockRandom []	BlockTrueRandom []
SeedRandom []	TrueRandomSequence []
SeedRandom [n]	TrueRandomSequence [n]
—	SetTrueRandomDataFile []
—	SetMaxTrueRandomSequenceLength []
RandomInteger []	TrueRandomInteger []
RandomInteger [{a, b}]	TrueRandomInteger [{a, b}]
RandomInteger [k]	—
RandomInteger [{a, b}, n]	TrueRandomInteger [{a, b}, n]
RandomInteger [{a, b}, {d ₁ , ..., d _k }]	TrueRandomInteger [{a, b}, {d ₁ , ..., d _k }]
RandomInteger [dist]	TrueRandomInteger [dist]
RandomReal []	TrueRandomReal []
RandomReal [{a, b}]	TrueRandomReal [{a, b}]
RandomReal [b]	TrueRandomReal [b]
RandomReal [{a, b}, n]	TrueRandomReal [{a, b}, n]
RandomReal [dist]	TrueRandomReal [dist]
RandomComplex []	TrueRandomComplex []
RandomComplex [{a, b}]	TrueRandomComplex [{a, b}]
RandomComplex [b]	TrueRandomComplex [b]
RandomComplex [{a, b}, n]	TrueRandomComplex [{a, b}, n]

Table 1: Comparison of standard functions provided by *Mathematica* with their counterparts implemented in RandFile package. In the above fName refers to a string with the path to the file with random data, dist refers to an object representing a probability distribution and the parameters used in functions implemented in RandFile follow the same pattern as the standard *Mathematica* functions.

Speed comparison of the three methods for producing random numbers in *Mathematica* is presented in Figure 1. One can notice that for a sample of size larger than 10^3 the QRNG on-line service is able to provide random numbers with a higher speed than the RandFile package. This is due to the fact that libQRNG library used to access this service allows downloading data in chunks, while *Mathematica* functions used to read random data from a file can read during a call one byte only. This is the case for **BinaryRead**, as well as **BinaryReadList**.

One can also notice that the use of SSD device, as it allows for the fastest read-out of data, increases the speed of random number generation. This illustrates the fact that the overhead of the read operations provided a significant contribution to total speed of the number generation.

4 Usage examples

In this section we provide some examples of the package functionality. As the main feature of the described package is the ability to harness file-based sources of random data, the package allows the actual manipulation of the quality of the provided randomness. Thanks to this, the user is able to observe the influence of quality of the provided data on the outcomes of the conducted simulations.

The assessment of the quality of the randomness sources is a challenging task [2]. In the following, however, we use the simple methodology implemented in the ENT program [18] and

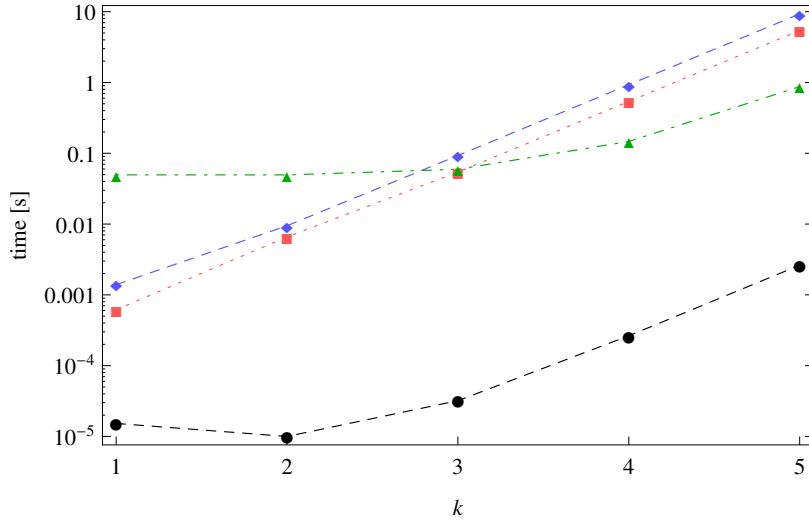


Figure 1: Comparison of the time required to generate a sample of 10^k real numbers for $k = 1, \dots, 5$ using pseudo-random generators provided by *Mathematica* (black circles), RandFile using standard hard drive (blue diamonds), RandFile using SSD hard drive (red squares) and the QRNG on-line generator via TRQS package using QRNG service (green triangles). The plots are based on the **AbsoluteTiming** function and the results were averaged over 5 experiments.

we will focus of the mean and the entropy of the input files used in the experiments.

4.1 Example 1: Value of π

As the first example we use the Monte Carlo method for calculating the value of π . The error in calculating this value can be used as an randomness indicator [18].

Using the functions provided by RandFile package one can easily implement a method for calculating the value of π using a given input file with random data. *Mathematica* function used in the following example is presented in Listing 1.

```
TrueRandomPi[fName_String] := With[{samplePoints = 5000},
  SetTrueRandomDataFile[fName];
  data = Table[TrueRandomReal[{0, 1}], {i, samplePoints}, {j, 2}];
  CloseTrueRandomDataFile[];

  dataCircle =
    Select[data, (0.5 - #[[1]])^2 + (0.5 - #[[2]])^2 <= (0.5)^2 &];
  complement = Complement[data, dataCircle];
  If[Length[complement] == 0, complement = {{1, 1}}];
  4*Length[dataCircle]/samplePoints
];
```

Listing 1: *Mathematica* code for the calculating value of π using Monte Carlo method based on data from a file.

The relative error of calculating the value of π using a random file is presented in Figure 2. Input files used in the experiment have been generated using **TrueRandomChoice** function with slightly biased weights. The entropy and the mean of the used files treated as sequences of bits are presented in Table 2.

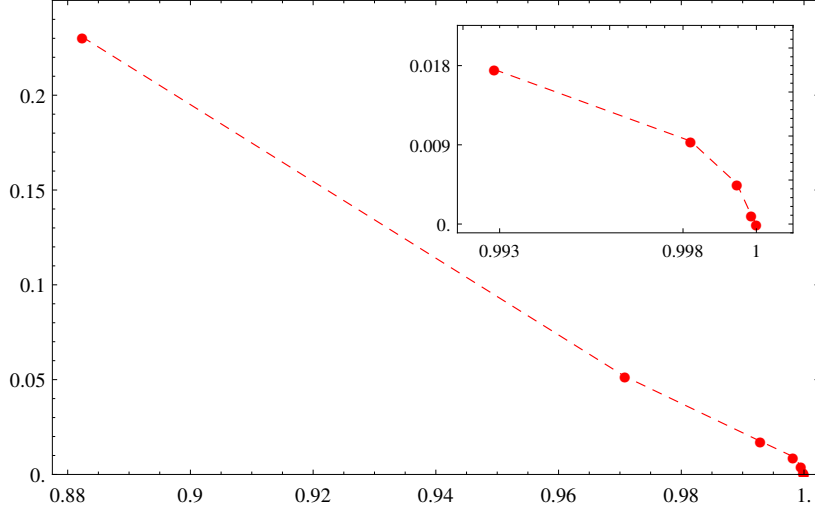


Figure 2: Relative error of calculating the value of π using the Monte Carlo method as a function of entropy of the input file used to generate samples. The value of π has been calculated using the function from Listing 1 for 7 sample files with different entropies. For the values of entropies close to 1, the values of the relative error are plotted in the inset.

bias	0.2	0.1	0.05	0.025	0.0125	0.00625	0.0
mean	0.6991	0.6002	0.5497	0.5249	0.5135	0.5069	0.5006
entropy	0.8823	0.9708	0.9928	0.9982	0.9994	0.9998	0.9999

Table 2: Parameters of the sample files used for calculating the value of π using Monte Carlo method. The entropy and the mean are calculated for the bit sequences obtained by using **TrueRandomChoice** function with weights biased towards 1. Each file contains 80KB of data.

Samplings obtained for selected input files are depicted in Figure 3. One can notice a clear pattern visible for the files with largely biased distribution of bits – Figures 3(a) and 3(b).

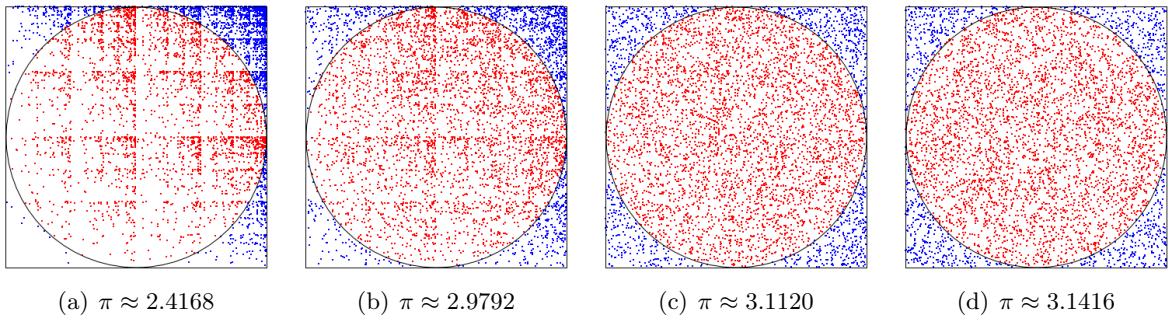


Figure 3: Representation of the samples used to calculate the value of π for samples with bias towards 1 equal 0.2 (a), 0.1 (b), 0.025 (c) and unbiased (d). See Table 2 for the parameters of the corresponding input files. Each sample contains 5000 points.

4.2 Example 2: Double slit experiment

To illustrate the influence of the bias in the source of random data on a physical experiment, we illustrate the usage of RandFile package for the purpose of simulating double slit experiment.

bias	0.2	0.1	0.05	0.025	0.0125	0.00625	0.0
mean	0.7000	0.6001	0.5501	0.5251	0.5126	0.5064	0.4999
entropy	0.8811	0.9708	0.9927	0.9981	0.9995	0.9998	0.9999

Table 3: Parameters of the sample files used for visualizing double slit experiment. The entropy and the mean are calculated for the bit sequences obtained by using **TrueRandomChoice** function with weights biased toward 1. Each file contains 2MB of data.

The parameters of the sample files used in this example are provided in Table 3. Simulation results are presented in Figure 4. Each plot represents relative intensities obtained for 101 detectors and using 1.5×10^5 events.

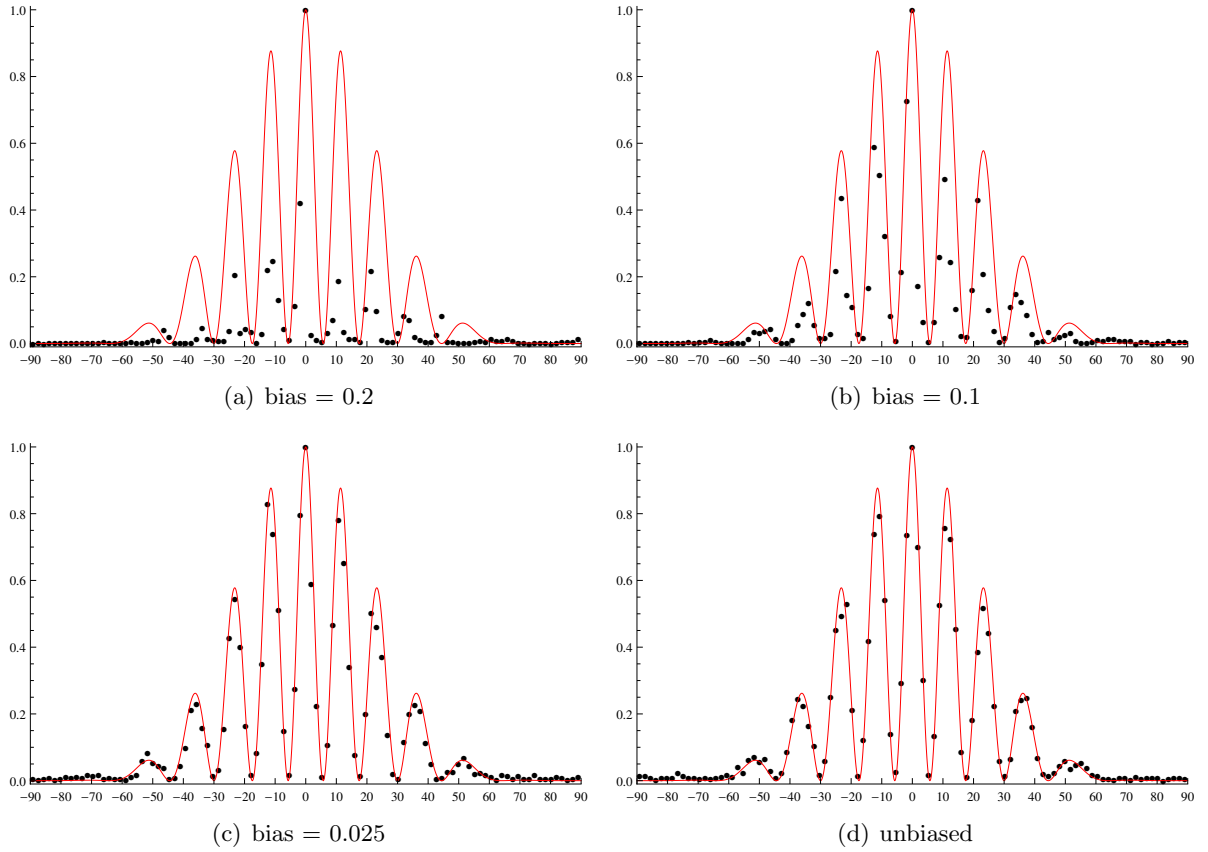


Figure 4: Relative intensity for angle $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$ calculated for simulation with 1.5×10^5 events. Solid red line represents the theoretical value.

The plots presented in Figure 4 visualize the influence of the quality of random numbers on the simulation results. In particular, for the input files with biased distribution of bits – Figures 4(a) and 4(b) – the simulation results are significantly different from the theoretical value.

5 Concluding remarks

The main contribution of this paper is the description of a package for *Mathematica* computing system which allows the production of random numbers using random data stored in files. This functionality enables the user to employ high-quality true random numbers in simulations. At the same time it allows the observation of the influence of the quality of random data provided on the outcomes of the numerical experiments.

The presented package uses local files for generating random numbers and it provides higher reliability in accessing random numbers during the simulation compared to true random number generators providing on-line user interface [1]. Moreover, there is no need for using any special hardware during the simulation and the data used can be obtained using various methods, including quantum true random number generators [17, 8] or methods based on other physical phenomena [19, 6].

The comparison of the efficiency of RandFile with the standard pseudo-random generator provided by *Mathematica* shows that the former is significantly faster. This can have a significant influence on performance of the simulations. On the other hand, the use of truly random data may be crucial for the correctness of the simulations results. However, the presented package does not require any external libraries and is based on *Mathematica* programming languages only.

Acknowledgements The presented was possible thanks to many interesting discussions with R. Heinen.

JAM would like to acknowledge financial support from the Polish Ministry of Science and Higher Education under the grant number IP2011 036371 and from the Polish National Science Centre under the research project DEC-2011/03/D/ST6/00413.

References

- [1] QRNG Service. Project developed by PicoQuant GmbH and the Nano-Optics group at the Department of Physics of Humboldt University. Web page <https://qrng.physik.hu-berlin.de/>.
- [2] L. Afflerbach. Criteria for the assessment of random number generators. *J. Comput. Appl. Math.*, 31(1):3 – 10, 1990.
- [3] H. Bauke and S. Mertens. Pseudo random coins show more heads than tails. *J. Stat. Phys.*, 114(3-4):1149–1169, 2004. arXiv:cond-mat/0307138.
- [4] A.M. Ferrenberg, D.P. Landau, and Y.J. Wong. Monte Carlo simulations: Hidden errors from “good” random number generators. *Phys. Rev. Lett.*, 69(23):3382–3384, 1992.
- [5] G.S. Fishman. *Monte Carlo. Concepts, Algorithms and Applications*. Springer Series in Operations Research. Springer-Verlag, New York, U.S.A., 1995.
- [6] M. Haahr. Random.org – True Random Number Service. Web page <http://www.random.org/>.
- [7] B. Hayes. The wheel of fortune. *Am. Scientist*, 81(2):114–118, March-April 1993.
- [8] ID Quantique SA. Quantis – True Random Number Generator Exploiting Quantum Physics. Product web page <http://www.idquantique.com/random-number-generators/products.html>.

- [9] I. Kanter, Y. Aviad, I. Reidler, E. Cohen, and M. Rosenbluh. An optical ultrafast random bit generator. *Nat. Photonics*, 4:58–61, 2010.
- [10] D. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical algorithms*. 3rd edition, 1997.
- [11] N. Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science*, (Special Issue 1987):125–130, 1987. Special Issue dedicated to Stanisław Ulam.
- [12] J.A. Miszczak. RandFile package for Mathematica.
- [13] J.A. Miszczak. Generating and using truly random quantum states in Mathematica. *Comput. Phys. Commun.*, 183(1):118–124, 2012. arXiv:1102.4598.
- [14] J.A. Miszczak. Employing online quantum random number generators for generating truly random quantum states in Mathematica. *Comput. Phys. Commun.*, 184(1):257258, 2013. arXiv:1208.3970.
- [15] S.K. Park and K.W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.
- [16] R. Schmied. Introduction to Computational Quantum Mechanics Fall Semester 2012, 2012. Lecture notes available on-line from <http://atom.physik.unibas.ch/teaching/CompQM.pdf>.
- [17] M. Wahl, M. Leifgen, M. Berlin, T. Rohlicke, H.-J. Rahn, and O. Benson. An ultrafast quantum random number generator with provably bounded output bias based on photon arrival time measurements. *Appl. Phys. Lett.*, 98(17):171105, 2011.
- [18] J. Walker. ENT – A Pseudorandom Number Sequence Test Program. Software available on-line from <http://www.fourmilab.ch/random/>.
- [19] J. Walker. HotBits: Genuine random numbers, generated by radioactive decay. Web page <http://www.fourmilab.ch/hotbits/>.